



BKM_DATS: Databázové systémy

4. Advanced SQL

Vlastislav Dohnal

Contents

- Advanced Aggregate Functions
- Analytic Functions
- Data Aggregation

- Sequences
- JSON Processing

Advanced Aggregate Functions

- General functions
 - `array_agg(expression)`
 - packs all input values into one array
- Statistical functions
 - `stddev_samp(expression)`
 - calculates the (sample) standard deviation over the values
 - `var_samp(expression)`
 - calculates the (sample) variance over the values
 - `corr(a,b)`
 - correlation coefficient between the two sets of values
 - `regr_slope(y,x)`
 - slope of the least-squares-fit linear function determined by the (x, y) pairs
 - `regr_intercept(y, x)`
 - y-intercept of the least-squares-fit linear equation determined by the (X, Y) pairs

Advanced Aggregate Functions

fraction \in $\langle 0;1 \rangle$

- (Inverse) Distribution functions
 - mode() WITHIN GROUP (ORDER BY expression)
 - returns the most frequent input value
 - choosing the first one arbitrarily if there are multiple equally-frequent results
 - percentile_cont(fraction) WITHIN GROUP (ORDER BY expression)
 - *continuous* percentile: returns a value corresponding to the specified fraction in the ordering,
 - interpolating between adjacent input items if needed
 - percentile_disc(fraction) WITHIN GROUP (ORDER BY expression)
 - *discrete* percentile: returns the first input value whose position in the ordering equals or exceeds the specified fraction

Advanced Aggregate Functions

- Hypothetical-set functions
 - `rank(value) WITHIN GROUP (ORDER BY expr)`
 - rank of the hypothetical value, with gaps for duplicate rows, over all values of `expr`.
 - `dense_rank(value) WITHIN GROUP (ORDER BY expr)`
 - rank of the hypothetical value, without gaps
 - `percent_rank(value) WITHIN GROUP (ORDER BY expr)`
 - relative rank of the hypothetical value, ranging from 0 to 1
 - `cume_dist(value) WITHIN GROUP (ORDER BY expr)`
 - relative rank of the hypothetical value, ranging from $1/N$ to 1

Analytic Functions

- provide the ability to perform calculations across sets of rows that are related to the current query row
- generally called *Window functions*
- <aggregate function>
OVER ([PARTITION BY <column list>]
ORDER BY <sort column list>
[<aggregation grouping>])
- E.g.,
SELECT ... ,
AVG(sales) OVER (PARTITION BY region
ORDER BY month ASC ROWS 2 PRECEDING), ...
FROM ...
 - moving average over 3 rows

Analytic Functions

- Ranking operators

- Row numbering is the most basic ranking function

- E.g.,

```
SELECT SalesOrderID , CustomerID ,  
       ROW_NUMBER() OVER (ORDER BY SalesOrderID )  
       as RunningCount  
FROM Sales WHERE SalesOrderID > 10000  
ORDER BY SalesOrderID
```

SalesOrderID	CustomerID	RunningCount
43659	543	1
43660	234	2
43661	143	3
43662	213	4
43663	312	5

Analytic Functions

- ROW_NUMBER doesn't consider tied values
 - Each 2 equal values get 2 different row numbers

SalesOrderID	RunningCount
43659	1
43659	2
43660	3
43661	4

- The behavior is nondeterministic
 - Each tied value could have its number switched!
- We need something deterministic
 - RANK() and DENSE_RANK()

Analytic Functions

- RANK and DENSE_RANK functions
 - Allow ranking items in a group
 - Syntax:
 - `RANK () OVER ([query_partition_clause] order_by_clause)`
 - `DENSE_RANK () OVER ([query_partition_clause] order_by_clause)`
 - DENSE_RANK
 - leaves no gaps in ranking sequence when there are ties
 - $\text{PERCENT_RANK} \leftrightarrow (\text{rank} - 1) / (\text{total rows} - 1)$
 - CUME_DIST - the cumulative distribution
 - the number of partition rows preceding (or peers with) the current row / total partition rows
 - The value ranges from $1/N$ to 1

Analytic Functions

□ Example

```
SELECT channel, calendar,  
       TO_CHAR(TRUNC(SUM(amount_sold), -6), '9,999,999') AS sales,  
       RANK() OVER (ORDER BY TRUNC(amount_sold, -6)) DESC) AS rank,  
       DENSE_RANK() OVER (ORDER BY TRUNC(SUM(amount_sold), -6)) DESC) AS  
dense_rank  
FROM sales, products  
... GROUP BY channel, calendar ORDER BY sales DESC
```

CHANNEL	CALENDAR	SALES	RANK	DENSE_RANK
Direct sales	02.2015	10,000	1	1
Direct sales	03.2015	9,000	2	2
Internet	02.2015	6,000	3	3
Internet	03.2015	6,000	3	3
Partners	03.2015	4,000	5	4

Analytic Functions

- Group ranking - RANK function can operate within groups: the rank gets reset whenever the group changes
 - A single query can contain more than one ranking function, each partitioning the data into different groups.
 - PARTITION BY clause

```
SELECT ... RANK() OVER (PARTITION BY channel ORDER BY SUM(amount_sold) DESC) AS rank_by_channel
```

CHANNEL	CALENDAR	SALES	RANK_BY_CHANNEL
Direct sales	02.2016	10,000	1
Direct sales	03.2016	9,000	2
Internet	02.2016	6,000	1
Internet	03.2016	6,000	1
Partners	03.2016	4,000	1

Analytic Functions

- NTILE splits a set into equal-sized groups
 - It divides an ordered partition into buckets and assigns a bucket number to each row in the partition
 - Buckets are calculated so that each bucket has exactly the same number of rows assigned to it or at most 1 row more than the others

```
SELECT ... NTILE(3) OVER (ORDER BY sales) NT_3 FROM ...
```

- NTILE(4) - quartile
- NTILE(100) - percentage

CHANNEL	CALENDAR	SALES	NT_3
Direct sales	02.2016	10,000	1
Direct sales	03.2016	9,000	1
Internet	02.2016	6,000	2
Internet	03.2016	6,000	2
Partners	03.2016	4,000	3

- Not a part of the SQL99 standard, but adopted by major vendors

Analytic Functions

- Obtain a value of a particular row of a *window frame* defined by window clause (PARTITION BY...)
 - `first_value(expression)`
 - `last_value(expression)`
 - `nth_value (expression)`

CHANNEL	CALENDAR	SALES	LOWEST_SALE
Direst sales	02.2016	10,000	4,000
Direst sales	03.2016	9,000	4,000
Internet	02.2016	6,000	4,000
Internet	03.2016	6,000	4,000
Partners	03.2016	4,000	4,000

```
SELECT ... FIRST_VALUE(sales) OVER (ORDER BY sales) AS lowest_sale
```

```
SELECT ... FIRST_VALUE(sales) OVER (PARTITION BY channel ORDER BY sales) AS lowest_sales
```

Analytic Functions

- Access to a row that comes before the current row at a specified physical offset with the current window frame (partition)
 - LAG(expression [,offset [,default_value]])
- ... after the current row
 - LEAD(expression [,offset [,default_value]])

CHANNEL	CALENDAR	SALES	PREV_SALE
Direst sales	02.2016	10,000	NULL
Direst sales	03.2016	9,000	10,000
Internet	02.2016	6,000	NULL
Internet	03.2016	6,000	6,000
Partners	03.2016	4,000	NULL

```
SELECT ... LAG(sales, 1) OVER (PARTITION BY channel ORDER BY calendar) AS prev_sales
```

Data Aggregations

- Used in GROUP BY clause instead of mere list of attributes
- ROLLUP (e1, e2, e3, ...)
 - represents the given list of expressions and all prefixes of the list including the empty list
- CUBE (e1, e2, e3, ...)
 - represents the given list and all of its possible subsets (i.e., the power set)
- GROUPING SETS ((e1,e2), (e4,e5), (e6), () ...)
 - rows are grouped separately by each specified grouping set
- Function to obtain which “GROUP BY” takes place
 - GROUPING(args...)
 - Integer bit mask indicating which arguments are not being included in the current grouping set

Data Aggregations

- Pivoting table for make and model over sales data
 - `SELECT make, model, sum(amount) FROM sales GROUP BY CUBE (make, model)`

	BMW	Mercedes	By model
SUV			
Sedan			
Sport			
By maker			
			SUM

Data Aggregations

- Example of CUBE on table of car sales (year, make, model, amount)
 - GROUP BY CUBE (year, make, model) calculates:

Aggregate

